Zerto

The State of Data Protection for Kubernetes

Version 2.0
April 2021

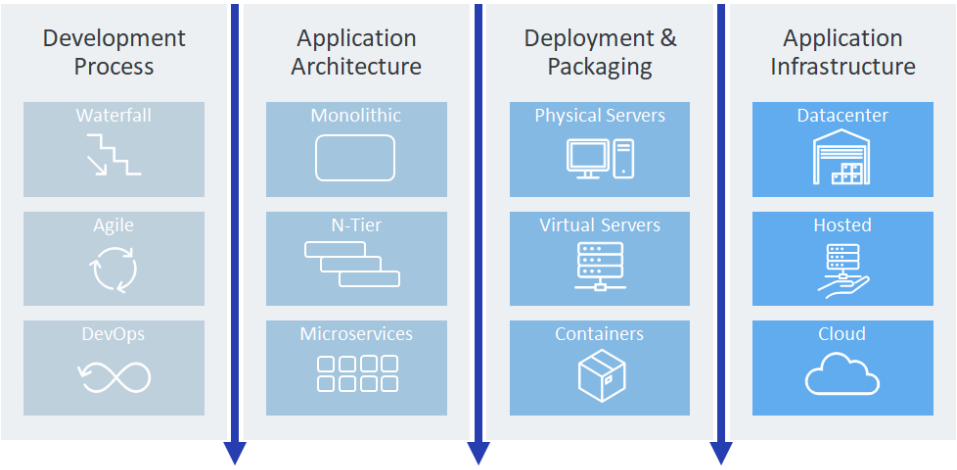
Zerto

Table of Contents

Container Adoption

Container technologies are becoming increasingly popular as organizations discover their technical and organizational advantages. Of these advantages, the major driving force behind containerization is a renewed focus on accelerating the software development life cycle, which has led to the adoption of several DevOps practices like cloud-native microservices.

Using containers, application developers are able to package small, focused code into independent, portable modules. Critically, these modules also include what's needed to run the code—only what's needed—making them lightweight and efficient.



Containers are one of the next-gen technologies and practices for developing and deploying applications.

ESG Research reports that Containers adoption is in full acceleration and in position to become the go-to choice for production deployment in the next 24 months. Container adoption is pervasive, with 67% of organizations already in production. Looking ahead to the next 24 months, the momentum continues, resulting in containers becoming the more widely used platform for production deployment, ahead of virtual machines. [3]

While these numbers forecast the industry's future, at the current moment, many IT organizations don't have containerization strategies, nor do they know how to manage containers at scale or protect and recover modern applications and persistent data. For some enterprises, bottom-up adoption means "containers just happen," and not unlike the cloud revolution that led to massive shadow IT, the container revolution could lead to uncontrolled grassroots use of containers, leaving groups to grapple with undesired effects.

So how do you give developers the freedom to use containers in their workflows while protecting the persistent data of their applications throughout the development life cycle and beyond?

Containerization's lightweight, modular approach has dramatically changed the way we build and run applications. Let's take a brief look at why containers—Kubernetes in particular—have amassed popularity among the developer and DevOps crowds from three different perspectives:

1. Developer agility
2. Decoupling of layers
3. Microservices

Container Advantages: Developer Agility

With vastly different needs, IT and DevOps are often siloed departments within an organization, perpetually leaving room for more consistent and frequent collaboration and alignment.

Challenged by the complexity of their environments, sometimes with legacy applications and aging infrastructure, IT departments are judged on quality of service: availability, performance, service level, response times, and more. Add these stresses to the high stakes of downtime or IT otherwise not working, and it's no wonder these groups want to control change management processes tightly.

On the other hand, development teams embrace change, constantly creating new features or functionality, fixing bugs, and eliminating security issues.

Developers are judged on output: velocity, productivity, time-to-market, quality of the code, and other similar factors. With their attention focused elsewhere, they aim to minimize time spent on deploying and configuring the infrastructure aspects of their applications. Instead, they want these components to be in place and to "just work" so they can successfully build and run applications—but despite this, thanks in large part to a frequent need to build stacks for each of their specific applications, development teams are playing a more prominent role in infrastructure as they define and configure what processes should look like.

This need for speed puts pressure on traditional IT. Not only were most IT systems never designed to cope with such agility, but the capital investment and complexity make them inert and hard to change—at least until the infrastructure nears the end of its economical life span. So how can a small IT infrastructure team deal with the massive scale and budgets to upgrade?

Public cloud computing has definitively tipped the scales, as most offerings excel in on-demand self-service with rapid elasticity and scalability—areas where in-house teams traditionally struggle. With this in mind, it's no surprise that developers have resorted to shadow IT, using cloud resources behind IT's back. This has enabled developers to care less about infrastructure by automating the toil and hiding complexity behind a thorough service offering.

Circling back, containers apply to and improve these situations by creating separation of concern: Containerization simultaneously allows IT to define base images without needing to know DevOps requirements and gives development teams the ability to take container images and use them to build their own stacks, adding middleware and other dependencies to the base image without involving infrastructure teams.

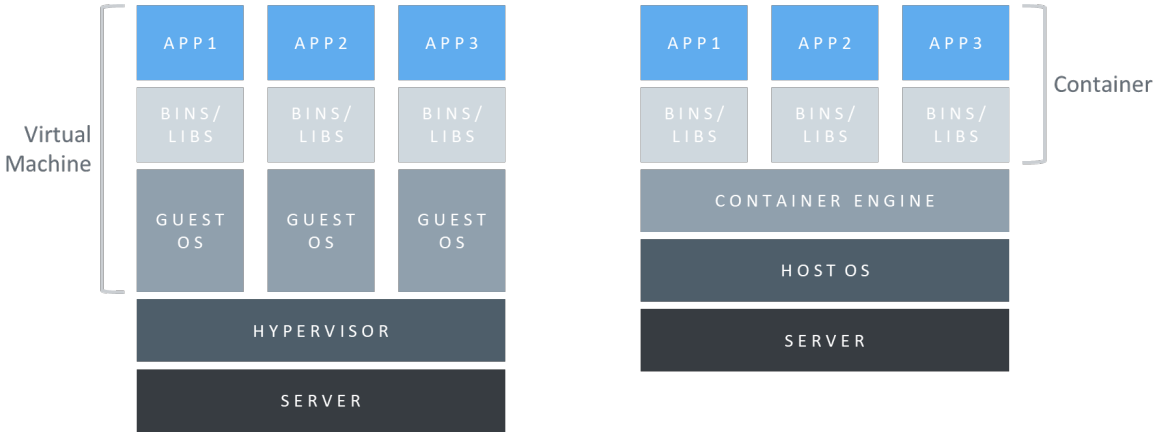
Container Advantages: Decoupling of Layers

Containers enable this separation of concern by using image layers. These layers can have different owners, and they can be stacked on top of each other. Compared to more traditional, physical storage, as well as VMs—in which storage layers are hard to separate—container images are a radical change in the way infrastructure is composed.

When it comes to VMs, the tight coupling of operating system, middleware, application binaries, application configuration, and application data makes building disk images a complex, slow process. For this reason, once a VM is built, configured, and running, it usually lives on for months or even years. VMs aren't typically rebuilt for upgrades to the operating system, middleware, database services, or any of their other major parts, either. Instead, they're upgraded in place.

At scale, the perennial nature of VMs begins to show its shortcomings. Over time, small differences in scope and initial deployment tend to occur, drifting the configuration, posing security risks, and building up technical debt with things like version conflicts, stability problems, and unused files. These often result in major headaches and frustration for development teams.

Without the ability to separate responsibilities, the task of VM construction is relegated mainly to IT operations—not developers—leading to an organizational disconnect that often causes developers to take matters into their own hands.



Unlike VMs, containers only include the minimum code and dependencies required in order to run, not an entire OS, and are thus more lightweight and portable.

Containerization simplifies and streamlines the build process. Every container comprises several immutable "layers" of a disk image, each defined by a base image and any relevant changes, such as those pertaining to additional software or configuration. These changes are codified in a text document, like a Dockerfile. A little higher up the stack, Kubernetes uses similar text files to define entire applications, potentially including many different container images.

Along with the ability to create images autonomously (based on IT-approved base container images), this high level of automation makes developers happy, allowing them to get what they need with little hassle. The immutable aspect of these containers turns what used to be a weeks-long process of requesting a VM with the right software into an off-the-shelf component with a container image that's ready for releases, deployments, testing, and other use cases.

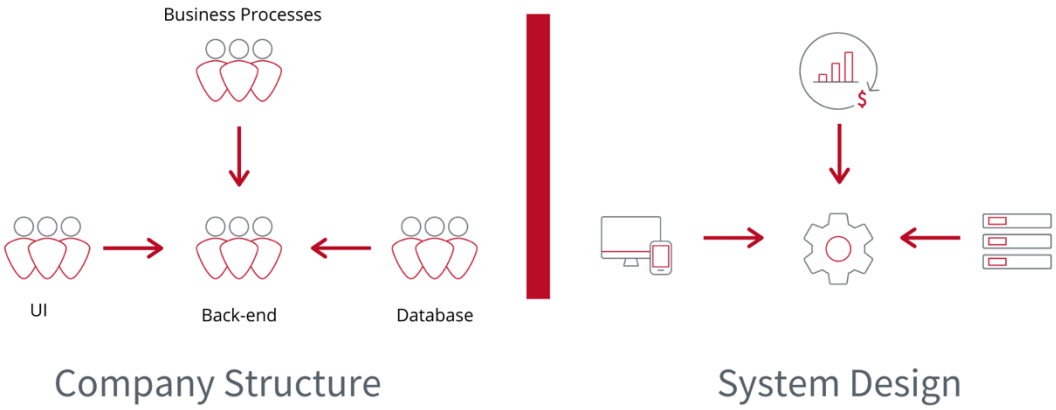
Containers are ideal in this sense. An organization's infrastructure team can deliver an artifact, a ready-to-go image for development teams to use and build on. These container image layers can then be reused across a wide number of applications; for example, a base Linux container being used across an entire enterprise Kubernetes estate.

With containers, infrastructure teams can make image updates at their own pace and according to their own processes. These updates are automatically picked up in the development pipeline, ensuring developer access to a secure, up-to-date version while maintaining any changes made in the additional layers—all without any effort on the part of the developers themselves. Containers also empower development teams to slim down the scope of their builds. Instead of pulling together massive images that contain everything from a bare operating system to their latest application build, developers only need to package their application into a container image. This lessens complexity, speeds up deployments, and reduces security risks.

Operational friction between development and infrastructure teams, an unnecessarily large and perennial infrastructure environment, and a lack of decoupling between layers all sharply contrast with how lean and agile software development truly is, making it no surprise that the traditional approach no longer works for modern processes.

Container Advantages: Cloud-Native Microservices

Many developers are choosing a more proactive approach to removing organizational complexity by breaking down their work into smaller pieces, creating more flow in the software development life cycle, removing manual processes, and taking ownership of all dependencies in the pipeline to bring code from local development all the way to production. Containers, microservices, and cloud-native application design facilitate this.



Container adoption presents the opportunity for system design to drive organizational structure rather than vice versa, as in Conway's Law.

A microservices architecture breaks down application components into their smallest possible units and assigns ownership of each to a multidisciplinary team. Loosely coupled within the larger service network that constitutes the application, these microservices are small and independent, allowing teams to operate separately from each other as they develop and release software. This reduces the time to market for new code and soothes the friction to which many developers were previously accustomed in the VM world.

In essence, coupling microservices with multidisciplinary teams represents a reverse of Conway's Law, which states, "Any organization that designs a system will inevitably produce a design whose structure is a copy of the organization's communication structure." Here, the opportunity lies in structuring communication around emerging developer practices.

Containerizing these microservices has become standard practice, leading to an explosion in both container usage and the total number of containers. Not unlike the VM sprawl that occurred 10 years ago, this container sprawl forces organizations to rethink their management strategies. Many decision-makers inevitably land on using Kubernetes, the de facto standard for container orchestration.

Kubernetes is an open-source platform that relies heavily on declarative configuration and automation to deploy and manage containerized applications. It has a vibrant ecosystem, much of which is also open source. Most notable in this context are the Container Storage Interface (CSI)—which provides storage services to Linux containers—and Helm, a package manager for finding, sharing, and using software built for Kubernetes.

Kubernetes is the orchestration layer that manages containers across a group of physical or virtual servers. Specifically designed to manage the ephemeral nature of thousands of containers spinning up, scaling up, and winding down, the platform manages versioning, exposes running services, handles storage considerations, and determines how containers should talk to each other over a given network. It also manages hardware failures and maintains container resilience.

Despite the platform's benefits, organizational challenges still run the risk of remaining unsolved, as they do with every technical platform. By separating responsibilities using multidisciplinary teams, microservices, and Kubernetes, groups run the risk of not implementing all the best practices from an IT operations or security perspective.

The Data Protection Challenges of Containers

One of the major areas where development and DevOps teams may fall short is data storage and data protection. Although Kubernetes includes some limited options for these, enterprises are finding gaps when it comes to true, end-to-end protection and resilience.

There are a number of reasons why data storage and data protection in container environments are challenging. At the core of this lies the simple fact that storage works very differently than it does with VMs—and compared to mature virtual environments, Kubernetes has fewer guardrails that ensure new workloads are configured correctly for data protection.

Let's look at how persistent storage differs between Kubernetes and virtualization platforms, such as VMware vSphere or Microsoft Hyper-V.

Containers & Pipelines

Immutability is foundational to the first major consideration: Immutable layers of the fully documented installation and configuration processes make up each container. Instead of merely capturing the end result—the container image—it makes more sense to protect the pipeline, the entire factory that produces the images, including all necessary configuration scripts (such as Dockerfiles and Kubernetes YAML files) and documentation.

Secondly, it's necessary to protect the systems that create the containers as part of the CI/CD pipeline, something that's often forgotten. Tools like code repositories, build servers (such as Jenkins), and artifact repositories that store containers and application releases fall into this category. By protecting these workloads, most of the "factory" that produces container images is kept safe.

Lastly, it's also important to capture the current state of running containers to avoid being forced to rebuild the entire deployment of many applications and even more containers in the event of a catastrophic failure.

Stateless vs. Stateful

After protecting the factory and the running state, there's still a major piece of the puzzle missing: protecting persistent application data.

In the earlier phases of container adoption, it was widely believed that containers were only suitable for stateless workloads and that storing any data in a container was impossible. We now know this is wrong: Both the underlying container runtime and Kubernetes fully support a diverse variety of workloads, including stateful applications.

While container images themselves are ephemeral—meaning that any file system changes are lost after the running container is deleted—there are plenty of options for adding stateful, persistent storage to a container. Even enterprise storage arrays in current use throughout on-premises datacenters can provide stateful storage to Kubernetes clusters.

In Kubernetes, persistent data is presented using the concept of volumes, each accessible to a container. At its core, a volume is simply a directory on the container host, featuring data within it. How that directory comes to be, the medium that backs it, and its contents are determined by the particular volume type used. PersistentVolumes are cluster-wide objects linked to the backing storage provider that make these resources available for consumption.
A PersistentVolumeClaim (PVC) is a storage consumption request from an application.

Organizational Alignment of Cloud Services

It's tempting to use a cloud storage service for object or file storage: It's quick, it's easy, and it's cheap (at least when you start out); however, a potential challenge arises when that cloud storage service remains separate, outside the control of those responsible for data protection. Invisible persistent storage resources lead to a risk of unprotected and insecure data as they operate without backup, disaster recovery, application mobility, and more.

When it comes to managing cloud storage, it's just as difficult to do it properly as it is with enterprise, on-premises storage. Organizations must ensure a consistent approach to accessing and managing cloud storage so developers can use the services they need while IT at-large maintains oversight, security, and overall responsibility.

Data Protection and Disaster Recovery

Looking back at the comparison between container storage and more traditional virtualization platforms, we see that container storage is perhaps even more complex because the data protection ecosystem is much less mature than that of its virtualization counterparts.

Several solutions exist in the virtualization space, tying into the hypervisor and storage platform to optimize data protection and disaster recovery for virtualized workloads. Given the major technical differences between VMs and containerized applications, simply porting over those solutions doesn't work: Container-native data protection requires a rewrite.

For example, one of the most powerful characteristics of Kubernetes is that everything is configured using declarative configuration code—i.e., infrastructure as code. By merely retrofitting a solution and forcing developers to use a separate interface to define data protection, you're not taking advantage of the new paradigm Kubernetes has to offer. Therefore, your organization is not reaping the platform's full benefits.

This works the other way around too. Why reinvent the wheel for containers when the wheel doesn't need reinventing? A good example is a journaling engine for granular point-in-time recovery.

Challenges Summary

We've learned that containers are becoming immensely popular because they enable developers to quickly and easily create the required infrastructure for their applications. With minimal hassle, containers can be built automatically and repeatedly, and their combination of both ephemeral and immutable characteristics make them a breeze to work with throughout production, minimizing technical debt and configuration drift over time.

Because containerization is still so new, many IT organizations don't fully understand how to properly, securely, and efficiently protect, migrate, and recover these modern applications and their persistent data—and without a good data protection strategy, they risk jeopardizing their data along with all the benefits they've gained by adopting containers in the first place.

So how do you give developers the freedom to use containers in their workflows while protecting the persistent data of their applications throughout the development life cycle and beyond? How do you make sure developers have no way to opt out of using data protection by making it a seamless, automatic, and transparent part of their workflows?

Zerto for Kubernetes

It's clear that containerized applications require disaster recovery, backup, and mobility—and selecting the right data protection solution makes a substantial difference in an organization's agility. Simply opting for non-native solutions from legacy backup and disaster recovery providers will only add time, resources, and barriers to application development and delivery.

To maximize their investment in the future, infrastructure and operations (IO) organizations must look for a platform that delivers the necessary availability and resilience without sacrificing the development speed of enterprise applications and services. They must be able to protect, recover, and move their containers without adding more steps, tools, and policies to DevOps organizations.

To provide both IT/IO and DevOps teams with the capabilities and outcomes they require to make the most out of their containerized applications and data, Zerto for Kubernetes introduces the following features:

Data Protection As Code

Reducing application downtime and data loss is a priority for any application—and since containers aren't immune to typical challenges like cyberattacks, accidental deletions, infrastructure failures, etc., it's crucial that organizations use a platform that delivers the ability to resume operations with little hassle when disaster strikes.

The native Zerto for Kubernetes solution drives a "data protection as code" strategy, integrating data protection and disaster recovery operations into the application development life cycle from day one. This means that applications are born protected. Using this approach, organizations can ensure the resilience of their applications without sacrificing the agility, speed, and scale of containerized applications.

Replication and Journaling Technology

Using Zerto for Kubernetes's replication and journaling technology gives you the freedom to simply rewind to a previous checkpoint, delivering a low recovery point objective (RPO). This nondisruptive approach offers more flexibility and availability than a traditional backup approach that uses snapshots—potentially hours behind production systems—and leaves gaps in data protection.

Zerto's continuous data protection (CDP) technology has long been the gold standard in the VM world. It should be no surprise that it's already considered the best option for containers as well.

Application-centric Protection and Mobility

True protection doesn't just include persistent data: It must provide the ability to protect, move, and recover a containerized application as one consistent entity, including all associated Kubernetes objects and metadata. Zerto for Kubernetes protects your applications' persistent volumes as well as all of their associated Kubernetes entities, such as deployments, StatefulSets, ConfigMaps, and services.

Avoid Vendor Lock-in

Zerto for Kubernetes is compatible with all enterprise Kubernetes platforms, allowing data to move to where each of your applications needs to run without locking you into a specific storage platform or cloud vendor. This means your persistent data is as mobile as the containers themselves.

Zerto for Kubernetes offers protection and support across leading platforms, such as:

- Microsoft Azure Kubernetes Service (AKS)
- Amazon Elastic Kubernetes Service (EKS)
- Google Kubernetes Engine (GKE)
- Red Hat OpenShift
- IBM Cloud Kubernetes Service

Regardless of the platform, being enterprise class means being technology agnostic and giving DevOps the flexibility and agility it needs. Data protection should follow suit.

Conclusion

Containers and Kubernetes are here to stay. Developers love the platform because it gives them the freedom they need to create, build, and run applications quickly; however, they share custody, as IT admins are responsible for staying in control and taking care of business continuity.

Zerto for Kubernetes helps solve the complex enterprise challenges of containerized applications, ensuring developers can continue to use Kubernetes and containers without the need to change their workflows while guaranteeing data protection and compliance coverage.

[1] https://451research.com/images/Marketing/press_releases/Application-container-market-will-reach-2-7bn-in-2020_final_graphic.pdf

[2] Red Hat Global Customer Tech Outlook 2019

[3] ESG Survey, Data Protection Trends and Strategies for Containers, September 2020

About Zerto

Zerto helps customers accelerate IT transformation by eliminating the risk and complexity of modernization and cloud adoption. By replacing multiple legacy solutions with a single IT Resilience Platform, Zerto is changing the way disaster recovery, data protection and cloud are managed. With enterprise scale, Zerto's software platform delivers continuous availability for an always-on customer experience while simplifying workload mobility to protect, recover and move applications freely across hybrid and multi-clouds. **www.zerto.com**

IS-15413